Lab 1: Introduction to SageMath

SageMath basics

This course makes extensive use of the mathematical program SageMath, which we will access through https://cocalc.com/. (If you want to install SageMath on your own computer, you can go to www.sagemath.org and follow the instructions there, but this is not necessary for the class. That version of SageMath will look slightly different.) SageMath is both an interactive program and a programming language. The purpose of this lab is to learn the basics of SageMath and use them to explore functions.

When you first open SageMath in your browser, you’ll see a page that looks like Figure 1. You can type commands into this worksheet, pressing Enter between lines. To execute the commands, press Shift+Enter.

![Figure 1: SageMath screenshot](image)

In this text, SageMath inputs and outputs will be represented in typewriter font. When several lines of input and output are given, the input will be preceded by a prompt, >>, even though no such prompt will appear on screen. (SageMath does have a command line mode with a prompt, but this is rarely used.) When you type in examples, don’t type in the prompt characters.
The way to learn SageMath (or any computer program, for that matter) is to interact with it, but the temptation to just read the text can be very strong. Thus, this basic introduction uses short bits of text followed by problems. Each problem consists of a task and Sage’s output when the task is performed correctly. Your job is to give SageMath the input necessary to produce the output shown.

If you type a number and evaluate the cell, SageMath will spit it back at you.

**Exercise 1.** Display the number 3.
3

**Exercise 2.** Display the number 1775.
1775

You can perform operations on numbers.

**Exercise 3.** What is $2^*2$? (The asterisk is standard notation for multiplication in most computer languages.)
4

**Exercise 4.** What is $10/4$? (The slash is standard notation for division.)
5/2

SageMath tries to make its answers as exact as possible, so it prefers fractions to decimals. To get an output in decimal form, place a decimal point (with or without a zero) after one or both numbers in the calculation.

**Exercise 5.** What is $1775/6$?
1775/6

**Exercise 6.** What is $1775/6$ in decimal form?
295.833333333333

SageMath uses the caret (^) to represent exponentiation. For example, $2^3$ means $2^3$.

**Exercise 7.** What is $2.5^6$?
244.140625000000
Memory and Variables

So far, we’ve done calculations but haven’t kept any of the results. In order to keep a value available for future use, you need to give it a name or label. These labels allow information to be retrieved from the computer’s memory.

You can think of memory as being like a desk on which the computer keeps the information it’s working with. You put information in memory every time you use the copy command on your computer. We say you are putting the text or whatever you copied on the computer’s clipboard.

Exercise 8. Type the sentence, “I’m learning programming!” (including the quotation marks) into a SageMath cell and evaluate the cell. A new cell will form. Copy and paste the sentence into the new cell. (HINT: Use the keyboard shortcuts Ctrl+C and Ctrl+V, or Cmd+C and Cmd+V on a Mac.) What sentence is on the computer’s clipboard?

Exercise 9. Type the sentence “What’s in memory?” into a SageMath cell and evaluate the cell.

1. Highlight and copy the sentence as in the previous exercise but don’t paste it anywhere. What sentence do you think is on the clipboard now?

2. Now, highlight and copy the sentence, “I’m learning programming!”. Go back to your blank cell and paste. Which sentence got pasted? What sentence is on the computer’s clipboard now?

The standard term for “giving a value a label” is “assigning the value to a variable”. You can do this using the = symbol. For example, to assign the value 10 to the variable h, enter h=10 and evaluate the cell. When you assign a value to a variable, SageMath displays nothing. You have to type in the name of the variable to see its value.

Exercise 10. Assign the value 3 to the variable mice and display it.

Exercise 11. Assign the value 7*94 to the variable prod and display it.

You can then use the variable in computations just like you would use a number. They are particularly useful for multi-step calculations.

```
>>a = 2*26
>>b = 29*12
>>val = a*b
>>val
18096
```
Exercise 12. Describe what each line in the above example does.

Exercise 13. Compute 67*45 and 93*442. Use variables to find their sum.

Exercise 14. In two steps, find the square root of the sum of 76.2 and 35.1.

Once a value has been assigned to a variable, that variable can be used anywhere in the worksheet.

Exercise 15. Without typing in the number 3, compute \(3^{20}\) and assign it to the variable bignum. Then, display bignum. HINT: What variable that you defined earlier has the value 3?

You can use a variable to change its own value. Try the example below.

```python
>>> dalmations=100
>>> dalmations=dalmations+1
>>> dalmations
101
```

Exercise 16. What is the value of dalmations in the first line?

Exercise 17. Briefly describe what the line `dalmations=dalmations+1` does.

Exercise 18. Assign the value 5 to the variable `num`. Then, add 2 to `num` and assign the result to `num`. Display `num`.

Exercise 19. What do the last two exercises have in common with Exercise 9?

**Debugging hint:** If a program you write behaves in a puzzling way, look at the values of the variables involved and how they change as the program runs.

The name of a variable can be basically anything that starts with a letter, although the only allowed punctuation mark is the underscore (_). SageMath won’t allow you to assign to a few critical Python keywords, although you’re not likely to want to use these words as variable names anyway. (To see a list of them, go to [http://en.wikipedia.org/wiki/Python_syntax_and_semantics#Keywords](http://en.wikipedia.org/wiki/Python_syntax_and_semantics#Keywords).) However, there is one common source of problems you should be aware of. For reasons you’ll learn about later, it’s a bad idea to assign numbers or other values to the variable `x`. Since it’s usually best to use descriptive variable names anyway (for example, `temperature` or `temp` rather than `t`), just get in the habit of never assigning values to `x` in SageMath.
Using built-in functions

SageMath has many built-in mathematical functions. For example, \texttt{abs} outputs the absolute value of a number, \texttt{sqrt} computes square roots and \texttt{ln} finds natural (base \textit{e}) logarithms. The input to the function goes inside parentheses. For example, to find the absolute value of -238, use \texttt{abs(-238)}.

\begin{exercise}
\textbf{Exercise 20.} Compute $\sqrt{37}$.

\texttt{6.08276253029822}
\end{exercise}

\begin{exercise}
\textbf{Exercise 21.} Compute $\ln(27.5)$.

\texttt{3.31418600467253}
\end{exercise}

Functions can be combined to make other functions. One way to do this is \textit{composition}: applying one function and then another. For example, if $f(x) = 2x + 1$ and $g(y) = \sqrt{y}$, then $g(f(x)) = \sqrt{2x + 1}$ and $f(g(y)) = 2\sqrt{y} + 1$. In SageMath, functions can be nested the same way, as in $\texttt{sqrt(abs(-4))}$.

\begin{exercise}
\textbf{Exercise 22.} Compute $\sqrt{|625 - 810|}$.

\texttt{13.6014705087354}
\end{exercise}

\begin{exercise}
\textbf{Exercise 23.} Compute $\sqrt{\ln(58.7)}$.

\texttt{3.74193931513816}
\end{exercise}

\begin{exercise}
\textbf{Exercise 24.} Redo the previous two exercises using variables instead of nesting.
\end{exercise}

SageMath has far too many built-in functions to keep track of, so it will allow you to assign a value to a variable that is also the name of a built-in function. For example, \texttt{ln=3} is perfectly valid SageMath code. However, executing this code will assign the name “\textit{ln}” to the number 3 instead of the function that takes natural logarithms. Thus, you will lose access to the built-in function \texttt{ln} until you restart the worksheet. To restart a worksheet, click the “Restart” button at the top of the worksheet. Then, reevaluate any code you need, making sure to get rid of the code that was overriding a built-in function.

\begin{exercise}
\textbf{Exercise 25.} Assign a value to the variable \texttt{abs}. Then, try to compute the absolute value of -2. When you finish, restart the worksheet. You shouldn’t need to reevaluate anything; just go on to the next exercise.
\end{exercise}

There is a bit of function-related vocabulary you should be aware of. Using a function is almost always referred to as \textit{calling} the function. Also, the inputs to a function are often referred to as its \textit{arguments}. For example, the command \texttt{sqrt(9)} calls the function \texttt{sqrt} with 9 as the argument.
Exercise 26. Call the function \texttt{ln} with the argument 35.0.
3.5534806148941

Reusing code

As useful as the built-in functions in SageMath are, you’ll often need to write your own. The general form of a SageMath function is:

\begin{verbatim}
def functionName(arguments):
    #Do stuff
    return output
\end{verbatim}

A very simple function could just use

\begin{verbatim}
def functionName(arguments):
    return output
\end{verbatim}

In this function skeleton, \texttt{def} is a keyword that marks the beginning of the function. The keyword \texttt{return} tells the function to give us the specified output and stop. Some functions, particularly those used for graphics and animation, don’t use \texttt{return}, but most others do.

Here is a simple function that takes a number and squares it.

\begin{verbatim}
def sqr(num):
    num2 = num^2 #Squares the number
    return num2 #Gives the output
\end{verbatim}

You could also write this in a shorter format.

\begin{verbatim}
def sqr(num):
    return num^2 #Squares the number and outputs it
\end{verbatim}

Here is another example: a function that multiplies its argument by 5 and raises it to the third power. Note that this calculation is done in one step.

\begin{verbatim}
def fivecube(n):
    result = (5*n)^3
    return result
\end{verbatim}

The body of a function must be indented relative to the top line. If you type the function directly in SageMath, the indentation will usually be inserted automatically. If necessary, use tabs to indent.
Exercise 27. Write a function that cubes its argument and multiplies the result by 19. Test your function on at least three different inputs. (HINT: To display the outputs of several commands on one line, separate the commands with semicolons.)

Exercise 28. Write a function that multiplies its argument by 5 and adds the square of the argument. Test your function on at least three different inputs.

Exercise 29. Write a function of your own choosing. Include comments explaining what each line does. Then, show the function’s output for several inputs.

The great virtue of functions is that they allow you to reuse code. Instead of copying and modifying code every time you want to use it, you can write a function just once. Then, you can call the function with different inputs, just like you can with built-in functions.

Plotting functions

One of the most useful things you can do with SageMath is plot functions. The basic plotting syntax is simple: \texttt{plot(f(x), (x,xmin,xmax))}. For example, to plot \( f(x) = 2x + 1 \) between -5 and 5, you would enter \texttt{plot(2*x+1, (x,-5,5))} and SageMath would output the graph in Figure 2.
Exercise 30. Plot $\sqrt{x}$ between 0 and 1000.

![Plot of $\sqrt{x}$ between 0 and 1000](image)

Exercise 31. Plot $|x|$ between -10 and 10.

![Plot of $|x|$ between -10 and 10](image)

The `plot` function has many useful additional options. For example, you can use the `color` option to change the color of the graph.

```python
>> plot(x^3, (x,-2,2), color="red")
```

You’ll encounter other plotting options in the examples that follow and in future labs. For now, notice that plotting options are arguments supplied to the `plot` function, so they **always** go inside the parentheses belonging to `plot`, **after** the function to be plotted and the range of values over which it should be plotted.

Exercise 32. Plot the function $f(x) = 5x^4$ for values of $x$ between -2 and 2 and make the plot green. NOTE: You do not need to define the function before plotting it.

In SageMath, you can use the `+` symbol to overlay plots. For example, `plot(x^2, (x,-2,2)) + plot(x^3, (x,0,2), color="purple")` yields the figure below.
Exercise 33. Overlay the graphs of two functions of your choice, making each graph a different color.

Interactive plots

One of the most powerful features in Sage is its ability to create interactive plots. Actually, such things don’t have to be plots and are commonly referred to as interactives. Here is an example of an interactive that lets you use a slider to manipulate the slope of a line. (Assigning a plot to a variable saves a graphics object that can be viewed later.)

```
@interact
def slopeLine(m=(-10,10)): #Gives the function name and slider range
    p=plot(m*x, (x,-5,5), ymin=-50, ymax=50) #Plots mx for a given value of m
    show(p) #Displays the plot
```

As you can see, the code for the interactive consists of the line `@interact`, followed by a Python function that performs the task you want. When defining the Python function for a slider-driven interactive, we specify the range its arguments can take. In the example, $m$ can range from -10 to 10.

Interactives can be much more powerful and complex than our simple example. You will use and program interactives throughout this course.
Exercise 34. Modify the example above so $m$ ranges from -2 to 3.

Exercise 35. What happens to the example if you try to plot the line without using show?

Exercise 36. Create an interactive that shows how a line with slope 1 changes as its y-intercept changes.

Exercise 37. Pick a function whose graph is not a line and write an interactive showing how that function responds to changes in a parameter. Include comments explaining what each line does.

Sometimes, it’s useful to constrain the slider in an interactive to particular values. This is often done by giving a step size as a third number when you specify the slider range. For example, the following interactive varies the slope of a line in increments of 0.25.

```python
@interact
def slopeLine(m=(-10,10,0.25)):  #Gives the function name and slider range
    p=plot(m*x, (x,-5,5), ymin=-50, ymax=50)  #Plots mx for a given value of m
    show(p)  #Displays the plot
```

Exercise 38. Modify your code from Exercise 37 so your parameter changes in increments of 0.1.